

|  |  |
| --- | --- |
| **NAME:** | SANIYA BANGARE |
| **UID NO:** | 2021300009 |
| **BATCH:** | A |
| **EXP NO:** | 7 |

**Aim:** To use Backtracking to solve the N Queen problem

# Theory:

* **Backtracking**

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time.

Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options.

Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won’t give any solution to that particular problem

**Algorithm: is safe**(chessboard,row, col, n)

1. for i = 0 to row - 1
2. if chessboard[i][col] == 1
3. return false
4. for i = row, j = col; i >= 0 and j >= 0; i--, j--
5. if chessboard[i][j] == 1
6. return false
7. for i = row, j = col; i >= 0 and j < n; i--, j++
8. if chessboard[i][j] == 1



1. return false
2. return true

**solve\_nqueen**(chessboard, row, n)

1. if row == n
2. print "Solution " ++ sol\_count ++ ":"
3. print\_chessboard(chessboard, n)
4. return
5. for i = 0 to n - 1
6. if is\_safe(chessboard, row, i, n)
7. chessboard[row][i] = 1
8. solve\_nqueen(chessboard, row + 1, n)
9. chessboard[row][i] = 0

# PROGRAM:

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

*//board index --> row no , borad[index] --> col no at which queen is placed*

int board[20], count;

int main()

{

    int n, i, j;

    void queen(int row, int n);

    printf(" - N Queens Problem Using Backtracking -");

    printf("\n\nEnter number of Queens:");

    scanf("%d", &n);

    queen(1, n);

    return 0;

}

*// function for printing the solution*

void print(int n)

{

    int i, j;

    printf("\n\nSolution %d:\n\n", ++count);

    for (i = 1; i <= n; ++i){

*//column indexes*

        printf("\t%d", i);

    }

    for (i = 1; i <= n; ++i)

    {

*//row indexes*

        printf("\n%d\t", i);

        for (j = 1; j <= n; ++j) *// for nxn board*

        {

            if (board[i] == j)

*// queen at i,j position*

                printf("Q\t");

            else

                printf(".\t"); *// empty slot*

        }

    }

    printf("\n\n");

}

*/\*funtion to check conflicts*

*If no conflict for desired postion returns 1 otherwise returns 0\*/*

int place(int row, int column)

{

    int i;

    for (i = 1; i <= row - 1; ++i)

    {

*// checking column cond.*

*// if board[current\_row] has value == current col .. not allowed*

        if (board[i] == column)

            return 0;

*// check diagonal*

        else if (abs(board[i] - column) == abs(i - row))

            return 0;

    }

    return 1; *// all cond met*

}

*// function --> if postion locked .. place queen .. move to next*

void queen(int row, int n)

{

    int column;

    for (column = 1; column <= n; ++column)

    {

*// printf("(%d-%d)\n" , row , column);*

        if (place(row, column))

        {

*// printf("yes\n");*

*// if all condition met .. place queen*

            board[row] = column;

*// all rows handled ... print final board config*

            if (row == n)

                print(n);

            else

*// one row done .. move to next*

                queen(row + 1, n);

        }

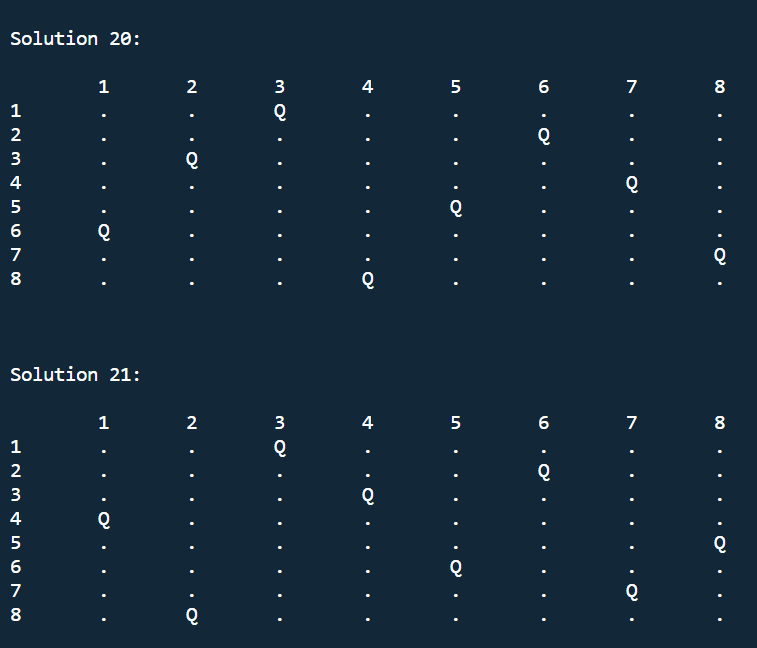
    }

}





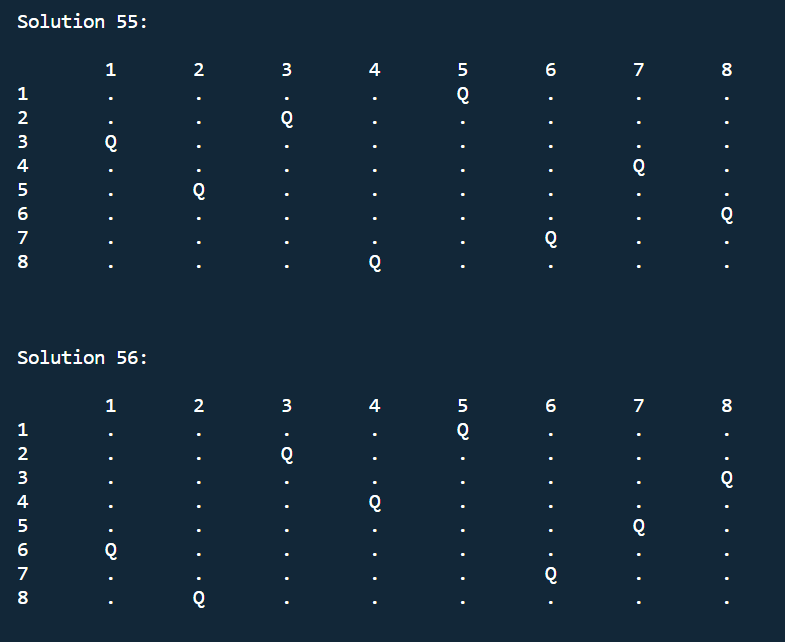
**Output:**





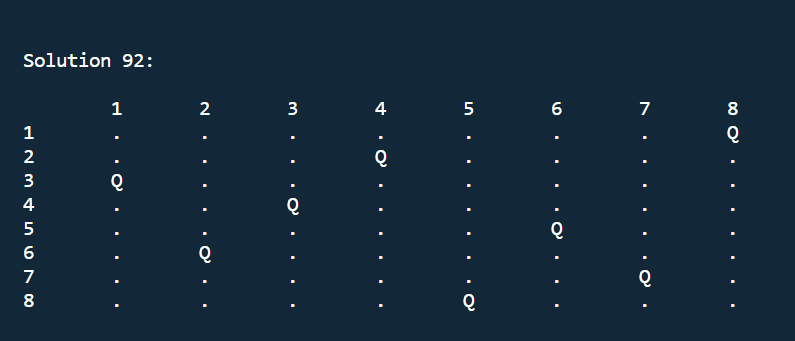


Middle solution:





2 Intermediate solution:



Total solution count: 92

# Observation:

* The n-queen problem has a significant computational complexity for large board sizes.
* The number of solutions increases rapidly with increasing board size.
* Backtracking algorithms like the one used in this implementation can efficiently find all solutions.

The time complexity is O(n!) as n queen is an combinatorics problem, we are just arranging n items.



# Conclusion:

In conclusion, the n-queen problem is a challenging and computationally complex problem. However, with backtracking algorithms, it is possible to efficiently find all solutions for small to moderate board sizes.